(4)

VLSI Memo No. 88-486
November 1988

GIRAPHE V3.3: A USER'S MANUAL WITH EXAMPLES

Robert M. Harris and Duane S. Boning

Abstract

GIRAPHE is a scientific plotting program that generates plots from files of tabular data. In addition to this basic capability, however, a great deal of effort has transformed the GIRAPHE program into a powerful data analysis utility. The principal capabilities and attributes of GIRAPHE are:

- Generation of graphs from files of tabular data,
- The appearance of plots is specified by the user through the GIRAPHE "command" file,
- Support of linear, logarithmic, and reciprocal axes,
- Shorthand notation in the "data" file to express complex collections of data.
- Expression evaluation to manipulate data prior to plotting.
- Support for a variety of graphics devices: GIRAPHE is capable of generating output for conventional graphics terminals (such as the VT241 or Tektronix terminals), workstation displays (running the X window system), and hard-copy printers or plotters (PostScript and HPGL output, for example).
- Support for incorporation of data directly into the command file and acceptance of data from the standard input. Thus, GIRAPHE can serve as a scientific plotting filter between data generating programs and display devices.
- A fully interpretive language, where commands are executed immediately. GIRAPHE can thus be used as a "shell," with the user entering commands directly from the keyboard.
- Operation under the Unix† (or Ultrix*) operating system.

† Unix is a trademark of the American Telephone and Telegraph Company.

* Ultrix is a trademark of the Digital Equipment Corporation.

Microsystems Research Center
Room 39-321
Massachusetts Institute of Technology
Cambridge Massachusetts 02139
Telephone (617) 253-8138

GIRAPHE V3.3

A User's Manual with Examples

Robert M. Harris and Duane S. Boning
MIT Microsystems Technology Laboratory

September 13, 1988

Contents

List of Tables

1 Introduction

GIRAPHE is a general purpose scientific plotting and data manipulation program. This introduction summarizes the capabilities of the program and describes the history behind GIRAPHE.

1.1 Capabilities

Fundamentally, GIRAPHE is a scientific plotting program that generates plots from files of tabular data. In addition to this basic capability, however, a great deal of effort has transformed the GIRAPHE program into a powerful data analysis (and thesis figure generating) utility. The principal capabilities and attributes of GIRAPHE are:

- Generation of graphs from files of tabular data.
- The appearance of plots is specified by the user through the GIRAPHE "command" file.
- Support of linear, logarithmic, and reciprocal axes.
- Shorthand notation in the "data" file to express complex collections of data.
- Expression evaluation to manipulate data prior to plotting.
- Support for a variety of graphics devices. GIRAPHE is capable of generating output for conventional graphics terminals (such as the VT241 or Tektronix terminals), workstation displays (running the X window system), and hardcopy printers or plotters (PostScript and HPGL output, for example).
- Support for incorporation of data directly into the command file and acceptance of data from the standard input. Thus, GIRAPHE can serve as a scientific plotting filter between data generating programs and display devices.
- A fully interpretive language, where commands are executed immediately. GIRAPHE can thus be used as a "shell," with the user entering commands directly from the keyboard.
- Operation under the Unix[†] (or Ultrix[*]) operating system.

[*] VMS and Ultrix are trademarks of the Digital Equipment Corporation.
[†] Unix is a trademark of the American Telephone and Telegraph Company.

1.2 Genealogy and Acknowledgments

The first version of GIRAPHE was made available for use within the MIT Microsystems Technology Laboratory in the Fall of 1986. The program has been built on the MFB device-independent graphics system from UC Berkeley, and uses parts of code from the MASTIF Workstation by Duane Boning. Major modifications and enhancements of the program were made by Bob Harris during the summer and fall of 1987. The program was exercised extensively by the authors, as well as by Jarvis Jacobs and Kurt Ware (whose bug reports have helped substantially in stabilizing the program). Subsequent enhancements and bug fixes during 1988 by both Harris and Boning have culminated in GIRAPHE V3.3; this version supersedes GIRAPHE V2.8, and is not command file compatible. The authors also thank Kurt Ware for extensive comments on this document, the first edition of the GIRAPHE manual.

1.3 Bug Reports

Bugs are anything in GIRAPHE that does not work as outlined in this manual. If something is difficult or inconvenient to do in GIRAPHE, that too is a problem we would like to hear about. Reports of bugs and additional comments are highly useful, not only in fixing the program, but in guiding future enhancements of the program. Submit reports to bug-giraphe@bacall.mit.edu; please be as specific as possible, and include (or tell where can be found) the command and data files that generate the bug.

1.4 Overview

The best way to begin using a program is to learn from known examples. For this reason, we begin this manual in Section 2 with a number of examples illustrating the basic use of the program. We begin with "minimal" GIRAPHE command and data files; subsequent examples introduce more powerful and complex features of the program.

Section 3 summarizing the invocation of the program follows the examples. In order to use GIRAPHE effectively, one must understand three main aspects of the program; the syntax and functionality provided by each is described in this manual. These three aspects are, first, the command file (which describes how the plot is to be generated), discussed in a reference format in Section 4; second, the data file (which describes the data to be used), discussed in Section 5, and finally the expression evaluator (which enables one to manipulate the data), discussed in Section 6.

2 Examples

In this section, we begin with a simple example summarizing the basic structure of the GIRAPHE command and data files, and showing the basic plotting capability of the program. In successive examples, we introduce the expression evaluator, the annotation and key commands, and finally the idea of a family of curves.

2.1 Example 1 – Basic Example

We begin with a very simple example involving only a single curve to be plotted from a single file of data. The GIRAPHE command file, named *example1.grp* looks like:

```
Title   Plot of Temperature vs Time
XLabel  Time (minutes)
YLabel  Temperature (Celsius)
Linear  xmin=0 xmax=20 ymin=900 ymax=1200
Read    File=example1.gdf xexp=#1 yexp=#2
Plot    Line=solid
End
```

while the data file, named *example1.gdf* appears as:

```
0       925
5       950
10      980
15      1025
20      1100
25      1250
```

The results produced by running GIRAPHE on this command file are shown in figure 1. In this figure, we see the single title line centered above the plot. We have specified linear axes for the plot, and have supplied the minimum and maximum values to be used on these axes. We then specify where the data is to be found with the "read" statement, and indicate that the "x" (or horizontal) values come from column 1 (#1) and the "y" (or vertical) values are in column 2. Once these values have been read into arrays within GIRAPHE, we indicate that we want a solid line drawn connecting these data points. Note that while the last data point (25, 1250) is not plotted, the line extending to that data point is drawn until clipped by the edge of the defined axes.

2.2 Example 2 – Expression Evaluator with Data File

The expression evaluator provides three additional capabilities to GIRAPHE.

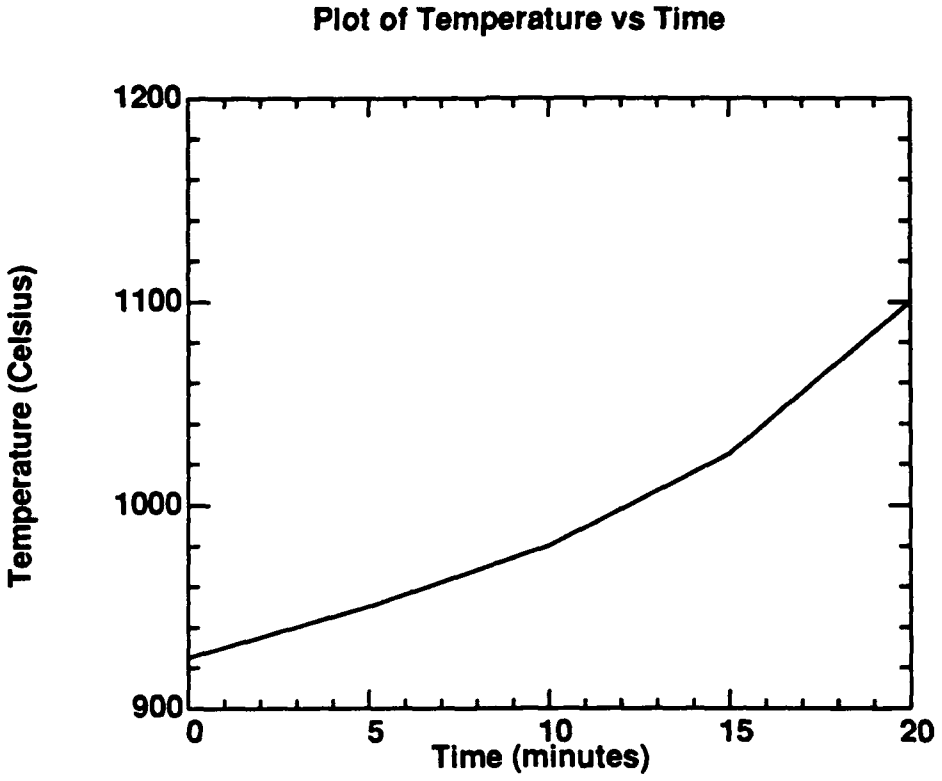

Figure 1: Plot produced for example1.

- The ability to refer to columns in the data file by name rather than by number. This increases ease of use, as well as documents what the values in the data file actually represent.
- The ability to manipulate the data from the file before plotting.
- The ability to "filter" data so that only those points meeting some criterion are used in the plot.

For instance, suppose we wanted to plot the temperature in degrees Kelvin, but only for times between 5 and 15 seconds (on the same axes as before). We label the columns in the data file, so that the file *example2.gdf* looks like:

```
.column time temp
.remark temperatures are in degrees celsius
0       925
5       950
10      980
15      1025
20      1100
25      1200
```

We modify the command file as below:

```
Title   Plot of Temperature vs Time
Linear  xmin=0 xmax=20 ymin=1100 ymax=1400
XLabel  Time (minutes)
YLabel  Temperature (Kelvin)
Read    File=example2.gdf xexp=time yexp=temp+273.15
+       filter=time>=5&&time<=15
Plot    Line=dashed Symbol=circle
End
```

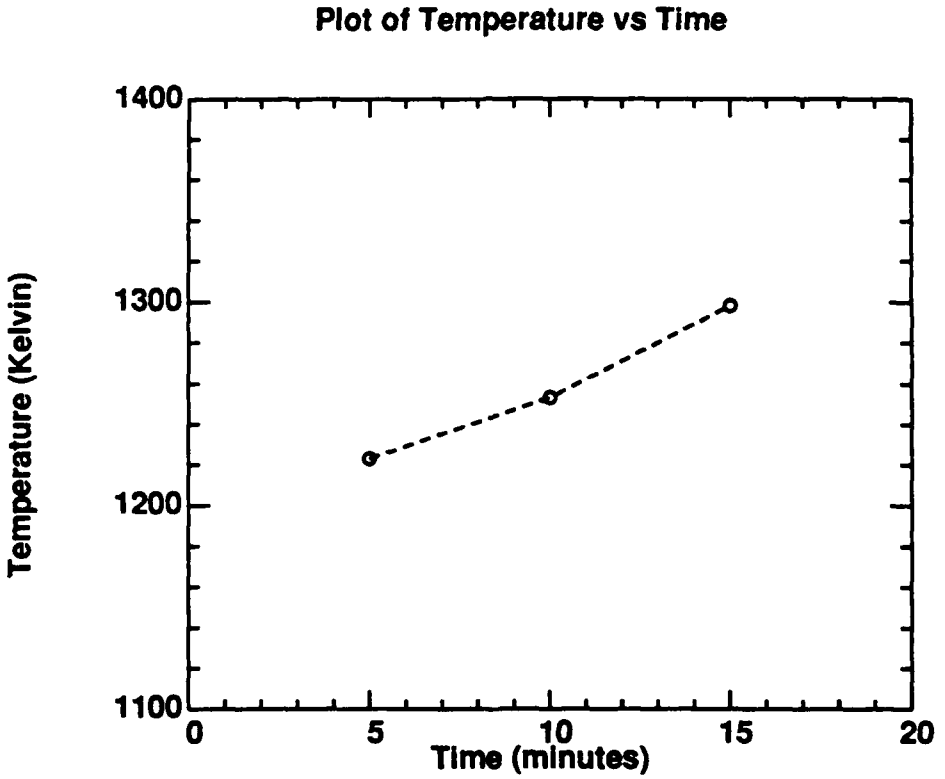

Figure 2: Plot produced for example 2.

The results of these command and data files are shown in figure 2. Here, we see that using `xexp=time` specifies indirectly that the first column of the file contains the x values, and `yexp=temp+273.15` causes the y value to be the second column added to the constant 273.15. In the continuation of the "read" statement (the "+" at the start of the next line indicates the continuation), we use a filter to pick out only those points where the time is between 5 and 15 seconds. In this example, we also request an additional output for the data. Here, each of the specific data points are drawn with a circle as the "symbol", and the points are connected with a dashed line.

2.3 Example 3 – Multiple Axes and Data Files

Sometimes the data one wishes to plot is not all contained within a single data file. One often also finds it useful to compare two different sets of values as a function of a third. This example illustrates both the use of an additional data file and multiple axes for data comparison. In this example, we also introduce the "Key" and "Legend" statements.

In addition to the data file of example2 (*example2.gdf*), we now add another data file, *example3.gdf*, appearing as:

```
.column time resistance
0     45.6
5     42.3
10    40.0
15    35.7
20    20.7
25    12.7
```

We modify the command file as below:

```
Title   Temperature and Sheet Resistance vs Time
Comment --- Set up axes and labels
Linear xmin=0 xmax=20 ymin=900 ymax=1200 ydelta=100 yfreq=2
YLabel Temperature (Celsius)
XLabel Time (minutes)
Axis    Right min=10 max=50
Label   right Text="Sheet Resistance (Ohms/square)"
Key     upper right
Comment -- Resistance vs Time
Read    File=example3.gdf xexp=time yexp=resistance
Legend  Resistance
Plot    Line=solid Symbol=square
Comment -- Temp vs Time
Use     yaxis=left
Read    File=example2.gdf xexp=time yexp=temp
Legend  Temperature
Plot    Line=dashed Symbol=circle
End
```

The results of this command file are shown in figure 3. The "Key" and "Legend" statements provide a mechanism for labeling individual lines drawn on the plot. The "Key" statement indicates that the key is to be placed in the upper right corner of the plot.

The first line plotted is based on the new data for resistance versus time. By default, the axes defined in the previous "Axis" statment are used, so that "x" will

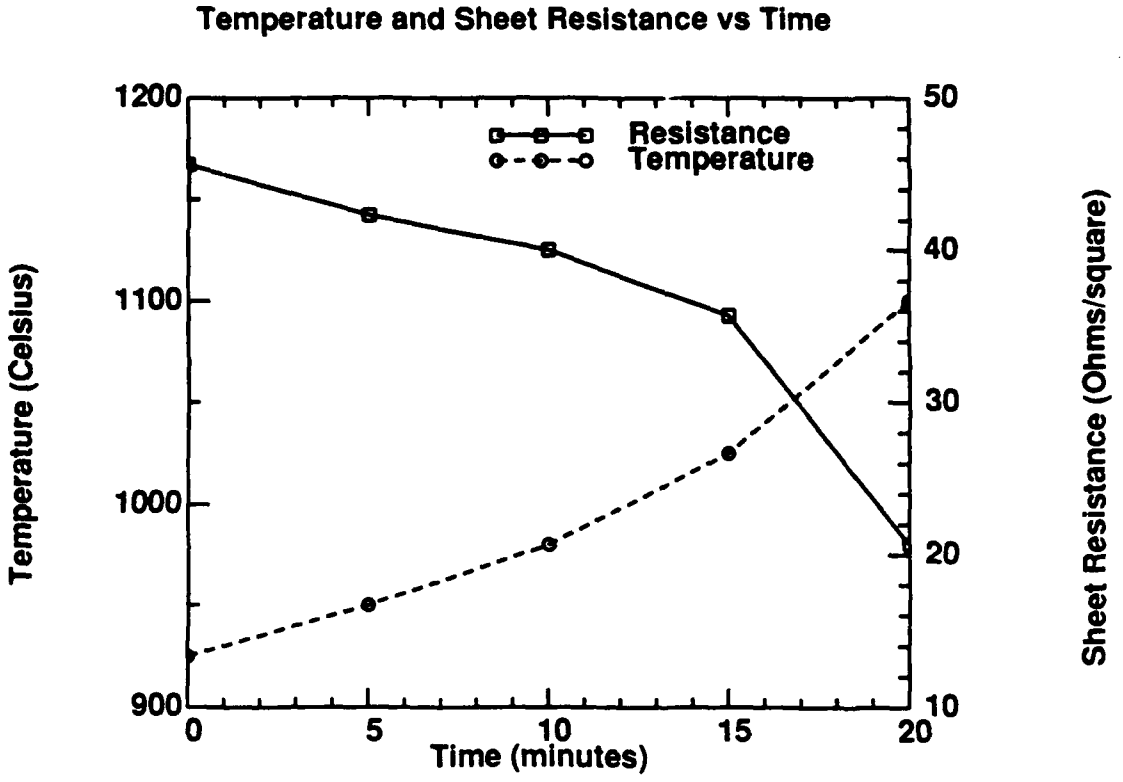

Figure 3: Plot produced for example 3.

be along the bottom (time), and "y" will be along the right axis (sheet resistance). The first "Legend" statement causes the specified text to be displayed in the key. Subsequent "Plot" statements, in addition to drawing data curves, will now also draw a line next to the legend text to complete the key entry.

To plot the second line, we want to switch the set of axes we are using. Here, we want to plot the temperature as the "y" value, and to use the tick marks along the left side. The "Use" statement associates the logical "x" and "y" with previously defined axes; in this case, we switch "y" to use the left axis.

Note also that we modified the appearance of the left tick marks slightly. Here, we specified `tic=100`, so that a major tick is placed every 100 units along the axis. We also use `ticfreq=2` so that we divide each major tic area into 2 regions, separated by minor tick marks.

2.4 Example 4 – Families of Data

Often one wants to plot several related curves. For example, we may want to plot several concentration versus depth curves, all parameterized by time. For example, suppose we have concentration versus depth data for three times: 0, 1200, and 4800 seconds. The concentrations are in inverse cubic centimeters (cm^{-3}) and the depths in centimeters (cm). We express this in the data file by using ".par" statements to

declare a parameter and ".set" statements to set the parameter value. The data file *example4.gdf* looks like:

```
.rem Times are in seconds, depths in cm, conc in cm**-3
.rem Define time as a parameter
.par time

.rem Define the first column as "depth" and the second as "conc"
.col depth conc

.rem Set the time for the first curve to be 0 sec.
.set time=0.0
0.0e-4 1e+15
1.0e-4 1e+15
2.0e-4 1e+15
3.0e-4 1e+15
4.0e-4 1e+15
5.0e-4 1e+15

.rem Set the time for the second curve to be 1200 sec.
.set time=1200.0
0.0e-4 1e+19
1.0e-4 1.57383e+18
2.0e-4 4.77727e+16
3.0e-4 1.22088e+15
4.0e-4 1.00015e+15
5.0e-4 1e+15

.rem Set the time for the third curve to be 4800 sec.
.set time=4800.0
0.0e-4 1e+19
1.0e-4 4.79552e+18
2.0e-4 1.57383e+18
3.0e-4 3.39915e+17
4.0e-4 4.77727e+16
5.0e-4 5.06911e+15

.rem This is the end of the file
.end
```

The command file, *example4.grp* we will use to plot this data is shown below:

```
Title Concentration vs. Depth
Title Infinite Source Diffusion
```

```
XLabel Depth (Microns)
YLabel Concentration (cm**-3)
SemiLog xmin=0.0 xmax=5.0 ymin=0.5e15 ymax=1e19
Key upper right
Comment Data file is in sec. and cm, we want min. and um.
Read file=example4.gdf Family=time/60.0 xexp=depth*1e4 yexp=conc
Legend Time = $#family$ min.
Plot line=solid color=foreground symbol=circle:square:triangle
End
```

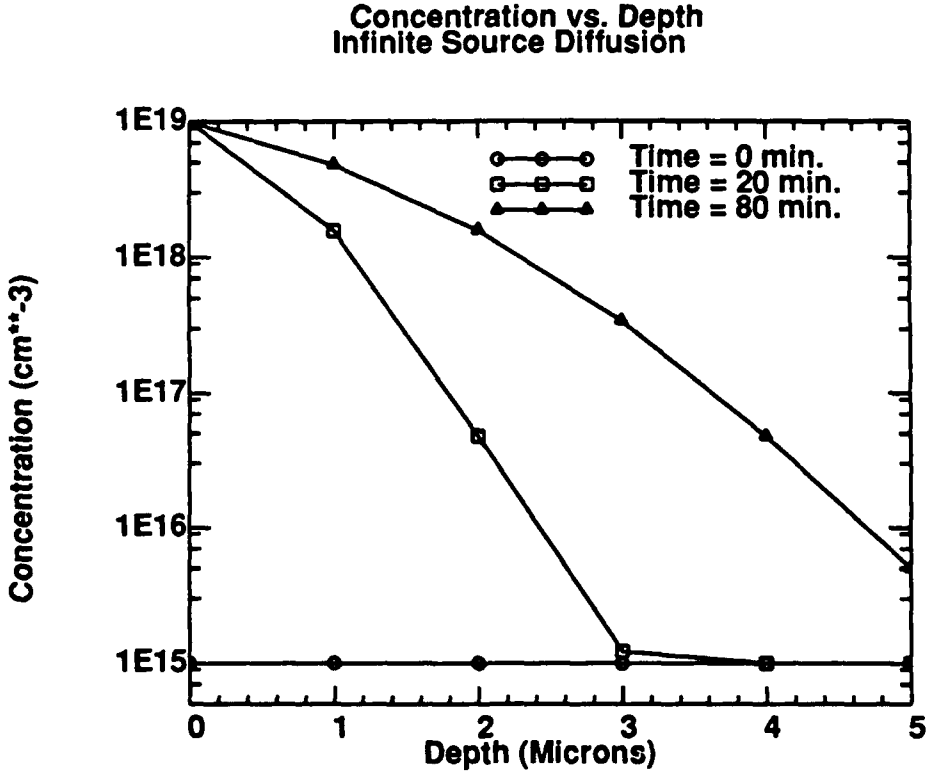

Figure 4: Plot produced in Example 4

The results of this command file are shown in Fig. 4. The "SemiLog" statement gives semi-logarithmic axes, i.e., a linear horizontal (or "x") axis and a logarithmic vertical (or "y") axis. A "LogLog" statement would have given two logarithmic axes.

The "Family" keyword in the "Read" statement tells GIRAPHE to produce a separate curve for each different "Family" value. Note that in the data file the times are in seconds and the depths are in centimeters. On the plot, however, the times are in minutes and the depths in microns. This is done by giving expressions as values for the "xexp" and "Family" keywords. The "xexp=depth*1e4" parameter

in the "Read" statement changes the depth from centimeters to microns and the "`Family=time/60.0`" keyword value changes the time from seconds to minutes.

The "Legend" statement in this example produces a slightly different legend for each curve in the family. The text between dollar signs (**$**) is not displayed as given but is first evaluated and then displayed. (If you actually want a dollar sign, "**$$**" will output a single dollar sign.) For the "Legend" statement the value of the special variable **#family** is the parameter of the current curve. There the text **$#family$** in the "Legend" statement is successively replaced by "0.0", "20.0", and "80.0." All statements that put text on the plot (the "Label", "XLabel", "YLabel", "Title", and "Legend" statements) can access the expression evaluator in this way. Just as we used one "Legend" statement for all three curves, we also use only one "Plot" statement for all three curves in this example. The value of the "Symbol" keyword in the "Plot" statement above is `circle:square:triangle`. This specifies that a circle, square, and triangle for each successive family plot are to be used as the plot symbol, where the colons are used to separate the successive values. If there are more curves than values we begin again at the beginning. This technique may also be used with the "Line", and "Fill" keywords. For example if we had specified "`Line=solid:dashed`" the first and third curves would have been plotted with a solid line and the second with a dashed line.

2.5 Other Examples

The examples in this section are available on-line for examination and test use with GIRAPHE. In addition to these, a number of additional test and example command files are usually available in the directory **/cad/src/giraphe3/examples**. Illustrating various features of GIRAPHE, these files include and demonstrate:

auto.grp	The axis autoscale feature.
bargraph.grp	Bargraph plot option.
contour.grp	Generation of contour plots.
iso.grp	Isometric scaling of axis limits.
key.grp	Use of keys and legends.
label.grp	Axis labeling (i.e., with days of the week).
peacock.grp	Tests color generation.
set.grp	Tests expression evaluator.

3 Running the program

GIRAPHE follows the Unix^{TM} convention for command invocation. The arguments to the GIRAPHE command are separated by spaces. Options are preceded by dashes ("-"). Certain options use the argument following the option (unless it is another option). For example, the sequence "-f blue" sets the foreground to the color blue. A synopsis of the GIRAPHE command as appears below is output when GIRAPHE is invoked with no arguments, or when invoked with an invalid option. In this usage summary, square brackets ("[ ]") denote optional arguments. For additional on-line documentation, consult the man page for "giraphe3".

giraphe3 [*giraphe-command-file*] [-t *mfb-device-type*]
[-d *device-name*] [-q [*queue-name*]] [-o [*output-file*]]
[-b *background-color*] [-f *foreground-color*]
[-l [*log-file*]] [-i] [-w] [-s] [-+] ,

The *command-file* (normally the first parameter of the GIRAPHE command) is the name of the GIRAPHE command file to be used (with an extension of ".grp" assumed). The command file provides information about the axes, titles, and how to find, manipulate, and present the actual data. The command file may direct GIRAPHE to read from *several* different files, each containing raw data points. Complete descriptions of the GIRAPHE command and data file formats appears in sections 4 and 5.

At most one command file may be given. If no command file is specified GIRAPHE will use *standard input* rather than a command file as the source of statements to execute. The program can therefore be used as a filter or run as an interactive "shell". (The -+ option described below is recommended for interactive use).

3.1 Output

GIRAPHE needs to know two things at invocation before it can produce graphical output. First, it needs to know the destination of the graphics output. Secondly, it needs to know what type of device is receiving the output in order to generate the correct graphics commands.

3.1.1 Output Destination

The destination of the GIRAPHE output may be selected by using one of the -d, -o, or -q options. The -o option is used to direct output to a file and specifies only the *immediate* and not the ultimate destination of the output. If an argument (which is not another dashed option) immediately follows the -o option, then the output file will have that as the file name. If the -o option has no argument, the name of the

output file will be the name of the command file, with the device type replacing the extension on the command file name. For instance, "giraphe3 foo.grp -t ps4 -o -s" will result in output to the file *foo.ps4*.

The -q and -d options are used to specify the ultimate destination of the output. The -d option is used to direct output directly to a device that will actually display the GIRAPHE plot. Under the X window system (using the "X" device type) the output will directed to the named display (i.e., "-o garbo:0"). Note that the -o and -d options are incompatible.

The -q option instructs GIRAPHE to send the output to the specified queue on plot completion. The GIRAPHE output is first placed in a file. The -o option can be used to set the name of the file used; if omitted, a temporary file is used and will be deleted once the file has been queued to the output device successfully. If a queue name is not given, the output will be queued to the default GIRAPHE queue (site dependent). Otherwise the queue name will be used for the "-P" option of the "lpr" command under Unix^{TM}.

If none of the above output directing options are specified, output will be directed to *standard output*, unless the "X" device type has been specified. In this case, the "DISPLAY" environment variable will be consulted to determine the device where the *mfb* emulation window will be opened. If the "DISPLAY" variable is not set, the -d must be used to specify the physical display device.

3.1.2 Output Device Type

The graphics display device type is specified with the -t option, followed by the *device type* . This device type should be a valid *mfb* device type or the special type "X" for use with the X Window System. If the device type is invalid, the message "Unknown terminal type!" will be displayed and the program terminated. Common *mfb* device types are shown in Table I.

If the device type is not explicitly specified, GIRAPHE will attempt to figure out the *mfb* device type from the ultimate destination of the graphics output (as specified by the -q or -d option). For the -q option, the queue name will be used to determine the device type. If output has been directed to a file (via the -o option) GIRAPHE cannot determine the device type since the ultimate destination of the file is unknown. If the destination has been specified by the -d option or is the *standard output*, GIRAPHE will use the environment variable "MFBTERM" (if set) as the device type. Failing that the "DISPLAY" environment variable is checked, and if set GIRAPHE will assume it is running under the X window system and use the device type "X". Finally, GIRAPHE will consult the information the operating system stores about the terminal. If GIRAPHE cannot determine the *mfb* device type, the error message "Unable to determine device type" will be displayed and the program terminated.

Table I: Common *mfb* Device Types

d3 VT241 terminal.

t25 Tektronix 4125 terminal.

X Creates an "mfb" emulation window. The user must specify the location of this window with the mouse. Note that automatic refresh of this window does *not* occur, so that moving, iconifying, or uncovering the window will result in destruction of the plot.

hc HPGL (Hewlett Packard Graphics Language) file output.

ps PostScript format file output. The **ps** format will print in the lower half of a page in a size appropriate for inclusion in a paper or report. Other device types with similar results include: **ps2** which will produce a full page PostScript plot, and **ps3** which produces a full page plot, with larger lines and text. Type **ps4** will generate a full horizontal rather than vertical page.

pp Unix plot(5) format file output. This file can then be passed through the standard Unix^{TM} plot filters. For instance, the "pp" file generated by GIRAPHE can be displayed on an ascii display using "plot -tcrt file.pp".

3.2 Colors

The background and foreground colors of the GIRAPHE plot are specified with the -b and -f options respectively. The default is a white foreground on a black background. GIRAPHE currently understands the following eight colors: "white", "black", "red", "blue", "green", "magenta", "cyan", and "yellow". Note that if the output device does not support software initialization of the color map (or this has been suppressed by the -i option described below), or does not have eight separately definable colors, then the actual color of the output may not be as desired. On most hard copy devices, specifying the color "black" will actually give the color white and *vice versa*. (It is usually better, then, to use the strings "foreground" and "background" rather than "white" and "black" when specifying colors inside GIRAPHE command files.)

Normally, GIRAPHE will attempt to initialize the output device's color map so that it can use all eight colors listed above. It cannot, however, reset the color map to its original state on completion. This leaves some terminals (especially VT241's) with particularly garish color schemes on completion. If the -i option is used, the program will not initialize the color map and the colors already defined for the device (if there are any) will be used instead.

3.3 Miscellaneous Options

3.3.1 Log File

The name of a log file for errors and verification (requested inside the GIRAPHE command file) may be specified using the -l option. If no argument is given to the -l option the log file name will be the command file name with the extension ".log". If -l is not specified the program will output the log information to the *standard output* unless the graphics output is already directed there. In this case the log information output will be suppressed.

3.3.2 Silent Output

On startup GIRAPHE will write the version number of the program, where the graphics output is going, and other information about files used to *standard output*. The -s option will force GIRAPHE to start silently with only graphics output written to *standard output* (if any). This is particularly useful when using GIRAPHE as a filter or running GIRAPHE as a background job.

3.3.3 Termination Wait

The -w option will cause GIRAPHE to wait for user response before terminating. This is particularly useful on devices that clear the screen as result of leaving graphics mode under *mfb*. If the graphics device has a pointing device, the plot will remain

until the user types a point (or hits a key) before terminating. If no pointing device is available, the program will wait for the user to enter the end of file character (ctrl-D under Unix^{TM}). Under the X window system (device type "X") the -w option is always in effect since the *mfb* window will be destroyed on termination of GIRAPHE.

3.3.4 Command Continuation

-+ Indicates that command lines may not be continued using a "+" character as the first character on a line. Normally, a "+" as the first character on a command line causes that command line to be a continuation of the previous textual line. This means, however, that the program must wait until the next line is available before executing the current one to see if the next line is a continuation of the first. When running GIRAPHE as an interactive shell this is undesirable, and the -+ option should be used. Note that lines may still be continued by using a backslash ("\") as the final character in a line.

3.4 Common Mistakes

The error messages produced by GIRAPHE are often a bit cryptic, and usually result from errors in the command or data files. Some of the more common errors are mentioned below.

- *Leading decimal:* Remember that the period (or decimal point) as the first character on a line in the data file introduces a data file directive (such as ".column"). A common mistake, then, is to start a number that is less than one with a decimal (".15"). The correct usage is either to use a leading zero ("0.15"), or to include at least one space or tab before the number begins.

- *giraphe rather than giraphe3:* The previous version of the program, GIRAPHE V2, is still in place on some systems. The version discussed in this manual in invoked with "giraphe3", while the previous version (retained for those poor users who have not upgraded) is invoked with "giraphe". If you run the program and a large number of syntax errors result, you might check that you have not run the wrong program.

- *Spaces in expression:* GIRAPHE uses spaces to break up the keyworded arguments in a command statement. A common error is to use a space when specifying a long expression in an attempt to make the expression more readable. This can be done, but the entire expression must then be enclosed by quotations ("), so that the space is included as part of the argument.

4 Command File Statements

The GIRAPHE command file consists of a sequence of statements, each statement on a new line. The purpose of the GIRAPHE command file is to give the necessary information (labeling, location of the data files, what axes to use, etc.) to generate the plot. The actual data for the plot is contained in the GIRAPHE data file which is described in the next section. Unless plus continuation has been disabled (by using the `-+` option described in the previous section) a statement may be continued to the following line with a plus sign ("+") as the first character of the continuation line. At all times, a line may be continued by ending it with a backslash ("\"). The immediately following line is then treated as a continuation.

GIRAPHE command file statements fall into four broad types: statements that manipulate axes, display text, manipulate data, and miscellaneous statements. The actual data for plotting is contained in the GIRAPHE data file described in Section 5. In the following subsections each of these statement types is described in general, followed by a detailed descriptions of individual commands.

Each description begins with a one or two line summary of what the statement does. This is followed by a synopsis of the statement showing the statement format. Because GIRAPHE is fully interpretive (i.e., each statement is executed immediately), the order that statements appear in the GIRAPHE command file is important. Constraints on each statement are given in the restrictions part of the command description. The statement's interaction with the evaluator is given next. The command description concludes with a detailed discussion of the command's effects.

Each GIRAPHE command statement uses the rest of the command line (following the statement name) to determine the arguments to the command statement. Some commands (principally those that display text) simply use the rest of the command line as the argument. This is indicated in the format synopsis below as "<*string*>". For example, the line

```
Title This is a title
```

will display "This is a title" as a title on the plot. Here embedded spaces do not require special treatment. The majority of GIRAPHE command statements, however, take a number of named arguments to which values are assigned using a "keyword equals value" syntax. That is, the name of the keyword is given, followed by an equals sign ("="), followed by the value. The values may be boolean (i.e. "true" or "false"), numeric, or a string of characters. In the format synopsis, this is represented by the name of the keyword, followed by an equals sign, followed by a type specifier enclosed in angle brackets (< >). Boolean values are represented by "<*b*>", numeric by "<*n*>", and character strings by "<*c*>". Note that the simple presence of the boolean argument in a statement indicates "true", so that "`Label left`" is equivalent to "`Label left=true`". To set a boolean to

false an up-arrow ("^") plus the name of the boolean may be used. For example, "`Label ^left`" is equivalent to "`Label left=false`". The keyword-value pairs themselves are separated by spaces or commas (","). To include a space or comma inside a value, the entire value must be enclosed in double quotes ("""). For example, "`Label text=word`" and "`Label text="with quotes"`" are correct, whereas "`Label text=no quotes`" will produce an error. The ordering of keywords within a command statement is not important; "`Axis left min=5 max=10`" and "`Axis max=10 min=5 left`" produce identical results. Command arguments may be optional (enclosed by "[ ]" in the format synopsis) or mandatory (not enclosed by square brackets). Occasionally only one parameter of several may be chosen; these "pick one" arguments are separated by "|". Finally, the default value for each keyword, if any, is shown in the format synopsis separated from the type specifier by a colon (":").

Parentheses are used for grouping in the format synopsis. For example, the format fragment:

[(min=<*n*> max=<*n*>) | auto=<*b*:false>]

means that the "min" and "max" keywords take numeric values and the "auto" keyword takes a boolean value. The square brackets enclosing all three keywords indicate that all three are optional. These are also pick-one keywords, i.e., either "auto" may be specified or "min" and "max" may be. Finally, we see that if "min" is specified, "max" must also be.

4.1 Axes

GIRAPHE allows four separate axes to be defined. The "left" and "right" axes are vertical axes and are displayed (if requested) on the left and right sides of the plot, respectively. The horizontal ("top" and "bottom") axes behave similarly. Three basic axis types are supported: linear (x), logarithmic ($\log_{10} x$), and reciprocal ($1/x$). For convenience in doing Arrhenius plots, four special reciprocal axes are provided suitable for use with temperature data in Kelvins, Rankins, degrees Centigrade, and degrees Fahrenheit. These axis types are summarized in Table 4.1.

Two basic statements are provided for axis handling in GIRAPHE. The **Axis** statement defines the limits and type of an axis, as well as the way the axis will be displayed on the plot. For example, an axis can be "omitted" (i.e., not displayed on the plot), or left unnumbered. The **Use** statement associates the logical x- and y-axes of the data with axes defined on the plot. This statement can be used to plot data using the left axis for the x-axis and the top axis for the y-axis, for instance. The are also four axis statements that serve as shorthand for specifying certain common types of axes. The **Linear**, **SemiLog**, **LogLog**, and **Arrhenius** statements give axes for linear, semi-log, log-log, and Arrhenius plots, respectively. All of these statements, discussed in detail immediately below, must come after any **Title** statements, and do not access the expression evaluator.

Table II: Summary of Axis Types

Axis Type	Axis Mapping
Linear	x
Logarithmic	$log_{10}x$
Reciprocal	$1/x$
Arrhenius Kelvin	$1000/x$
Arrhenius Rankin	$1000/x$
Arrhenius Celsius	$1000/(x+273.17)$
Arrhenius Fahrenheit	$1000/(x+459.7)$

4.1.1 The AXIS statement

Summary: The Axis statement defines an axis and controls its appearance on the plot.

Format: **Axis** [left=<*b*:false>] [right=<*b*:false>]
[top=<*b*:false>] [bottom=<*b*:false>]
[(min=<*n*> max=<*n*>) | auto=<*b*:false>]
[start=<*n*>] [delta=<*n*>] [freq=<*n*>]
[length=<*n*:0.015>] [factor=<*n*:2.0>]
[color=<*n*:foreground>]
[linear=<*b*> | logarithmic=<*b*> |
arrhenius=<*b*> [celsius=<*b*> | kelvin=<*b*> |
fahrenheit=<*b*> | rankin=<*b*>]]
[omit=<*b*:false>] [frame=<*b*>] [number=<*b*>] [tick=<*b*>]
[isotropic=<*b*:false>]

Restrictions: The **Axis** statement must occur after any **Title** statements and before any **Plot**, **Annotate**, or **Key** statement. Automatic scaling (the "auto" keyword) assumes that data has have been read in by a previous **Read** statement.

Evaluator: No access to the expression evaluator through this statement.

Description:

The **Axis** statements defines axes along the top, bottom, left, or right side of the plot, and immediately displays the defined axes on the plot. In order to define an axis, the location, axis type, and axis limits are required. The location is specified using the boolean keywords "left", "right", "top", or "bottom". Note that several axes may be defined by a single **Axis** statement; each of these axes will be identical except for location. The axis type may be specified using the

boolean keywords "linear", "logarithmic", or "reciprocal". The reciprocal axis type can be further qualified with the boolean keywords "kelvin", "rankin", "celsius", or "fahrenheit" corresponding to the special temperature reciprocal axes. These keywords correspond to the axis types in Table 4.1. If no axis type is given, the axis type of the opposite axis (right or left, top or bottom) will be used. Axis limits may be specified using the "min" and "max" keywords or the "auto" keyword. The "min" and "max" keyword allow the user to specify the axis limits directly. The value of the "min" keyword is the value of the left- or bottom-most limit of the axis; this value is normally the minimum value of the axis. Likewise, the value of the "max" keyword specifies the right- or top-most limit of the axis. The "auto" keyword will cause GIRAPHE to set the limits of the axes based on the data read in by the last **Read** statement. The limits are chosen so that all the data points will be inside the axis limits. If no limits are specified, GIRAPHE will use the limits of the opposite axis. If the opposite axis is a different axis type then GIRAPHE will chose limits so that the mapped limits (the result of applying the mapping function) on the two axes are the same. For example, the statements

```
Axis left log  min=10.0 max=1e5
Axis right linear
```

will define the right-hand axis to be a linear axis with the bottom-most limit to be 1.0 and the top-most limit to be 5.0.

The rest of the keywords control the way the axis will be displayed on the plot. The axis as displayed consists of three parts: the frame (the line running the length of the axis), major and minor ticks, and the numbering at the major ticks. The boolean keywords "frame", "tick", and "number" specify the presence or absence of the corresponding part of the axis. The default value of the keyword "frame", "tick", and "number" is "true". That is, GIRAPHE will display the whole axis (drawing the axis frame and tick marks and number major ticks) unless specifically requested not to by setting the appropriate keyword "false". If the boolean keyword "omit" is specified, however, defaults for the these keywords is "false" and only those parts of the axis that are explicitly requested are displayed. By default, the axis will be displayed in the foreground color. This may be overridden using the "color" keyword.

The position and appearance of the major and minor tick marks on the axis are controlled using the "delta", "freq", "start", "length", and "factor" keywords. The "start" keyword indicates the value to begin labeling as a major tick. The "delta" keyword is the increment between major ticks. For logarithmic-type axes this increment is a multiplicative increment. For all other types this increment is additive. The "freq" keyword gives the number of minor tick spaces between each major tick (i.e., `freq=5` displays four minor ticks, dividing the region between major ticks in five parts). The "length" keyword gives the length of the minor tick as a fraction of the plot area on the screen; the major tick length is this length

multiplied by the value of the "factor" keyword. Under normal conditions none of these keywords need be specified, the program places major and minor tick marks based on the axis limits.

4.1.2 The USE statement

Summary: The Use statement associates the logical x- and y-axes with the left, right, top, or bottom plot axes.

Format: Use [xaxis=<*c*>] [yaxis=<*c*>]

Restrictions: The Use statement can associate a logical axis only with a previously defined plot axis.

Evaluator: No access to the expression evaluator through this statement.

Description:

By default, the x-axis for actual plotting is the most recently defined horizontal (bottom or top) axis, and the y-axis is the most recently defined vertical (left or right) axis. The keyword "xaxis" may have a value of either "left", "bottom", "right", or "top" and is used to associate the x-axis with one of the previously defined plot axes. The "yaxis" keyword acts similarly for the y-axis. The x-axis may be associated with a vertical axis and the y-axis with a horizontal axis. However, the x- and y-axes may not both be associated with horizontal axes or both with vertical axes.

4.1.3 The LINEAR statement

Summary: A shorthand for defining axes for a linear plot.

Format: **Linear** [xmin=<*n*:0.0>] [xmax=<*n*:100.0>]
[ymin=<*n*:0.0>] [ymax=<*n*:100.0>]
[xdelta=<*n*>] [ydelta=<*n*>]
[xfreq=<*n*>] [yfreq=<*n*>]
[xstart=<*n*>] [ystart=<*n*>]
[length=<*n*:0.015>] [factor=<*n*:2.0>] [color=<*c*:fore>]
[omit=<*b*:false>] [frame=<*b*>]
[number=<*b*>] [tick=<*b*>]
[isotropic=<*b*:false>] [auto=<*b*:false>]

Restrictions: The **Linear** statement must occur after any **Title** statements and before any **Plot**, **Annotate**, or **Key** statement. Automatic scaling (the "auto" keyword) assumes that data has have been read in by a previous **Read** statement.

Evaluator: No access to the expression evaluator through this statement

Description:

This is a short hand statement for specifying a linear plot (where both left and bottom axes are linear). Here both the x (horizontal) and y (vertical) axis limits must be specified. Note that a decreasing axis is possible, as in `linear xmin=100 xmax=0`. The isotropic parameter indicates that one unit in x is equal to one unit in y, and that the limits of the axis will be scaled to give a one-to-one aspect ratio on the display device. Other parameters are analogous to those in the more general **Axis** statement. The **Linear** statement is equivalent to two **Axis** statements. The following line, for example,

```
Linear xmin=20.0 xmax=40.0 ymin=5.0 ymax=95.0 ystart=10.0
```

is equivalent to the following two **Axis** statements:

```
Axis linear left min=5.0 max=95.0 start=10.0
Axis linear bottom min=20.0 max=40.0
```

4.1.4 The SEMILOG statement

Summary: A shorthand to define axes for a semi-log plot.

Format: **SemiLog** [xmin=<*n*:0.0>] [xmax=<*n*:100.0>]
[ymin=<*n*:0.0>] [ymax=<*n*:100.0>]
[xdelta=<*n*>] [ydelta=<*n*>]
[xfreq=<*n*>] [yfreq=<*n*>]
[xstart=<*n*>] [ystart=<*n*>]
[length=<*n*:0.015>] [factor=<*n*:2.0>] [color=<*c*:fore>]
[omit=<*b*:false>] [frame=<*b*>]
[number=<*b*>] [tick=<*b*>]
[isotropic=<*b*:false>] [auto=<*b*:false>]

Restrictions: The **SemiLog** statement must occur after any **Title** statements and before any **Plot**, **Annotate**, or **Key** statement. Automatic scaling (the "auto" keyword) assumes that data has have been read in by a previous **Read** statement.

Evaluator: No access to the expression evaluator through this statement.

Description:

This is a short hand statement for specifying a semi-logarithmic plot, where the bottom axis is linear and the left axis is logarithmic. Other parameters are analogous to those in the more general **Axis** statement. The **SemiLog** statement is equivalent to two **Axis** statements. The following line, for example,

```
SemiLog xmin=-0.4 xmax=2.0 ymin=1e14 ymin=1e21 xstart=0.0 xdelta=1.0
```

is equivalent to the following **Axis** statements.

```
Axis linear bottom min=-0.4 max=2.0 start=0.0 delta=1.0
Axis log left min=1e14 max=1e12
```

4.1.5 The LOGLOG statement

Summary: A shorthand for defining axes for a log-log plot.

Format: **LogLog** [xmin=<*n*:0.0>] [xmax=<*n*:100.0>]
[ymin=<*n*:0.0>] [ymax=<*n*:100.0>]
[xdelta=<*n*>] [ydelta=<*n*>]
[xfreq=<*n*>] [yfreq=<*n*>]
[xstart=<*n*>] [ystart=<*n*>]
[length=<*n*:0.015>] [factor=<*n*:2.0>] [color=<*c*:fore>]
[omit=<*b*:false>] [frame=<*b*>]
[number=<*b*>] [tick=<*b*>]
[isotropic=<*b*:false>] [auto=<*b*:false>]

Restrictions: The **LogLog** statement must occur after any **Title** statements and before any **Plot**, **Annotate**, or **Key** statement. Automatic scaling (the "auto" keyword) assumes that data has have been read in by a previous **Read** statement.

Evaluator: No access to the expression evaluator through this statement.

Description:

This is a short hand statement for specifying a logarithmic plot, where both the left and bottom axes are logarithmic. The **Loglog** statement is equivalent to two **Axis** statements. The following line, for example,

```
Linear xmin=0.1 xmax=100.0 ymin=0.1 ymax=100.0 isotropic
```

is equivalent to the following **Axis** statements:

```
Axis log left min=0.1 max=100.0 isotropic
Axis log bottom min=0.1 max=100.0 isotropic
```

4.1.6 The ARRHENIUS statement

Summary: A shorthand for defining axes for an Arrhenius plot. The temperature may be given in units of Kelvin, Rankin, degrees Celsius, or degrees Fahrenheit.

Format: **Arrhenius** [tmin=<*n*:900.0>] [tmax=<*n*:1200.0>]
[ymin=<*n*:1.0>] [ymax=<*n*:100.0>]
[tdelta=<*n*>] [ydelta=<*n*>]
[tfreq=<*n*>] [yfreq=<*n*>]
[tstart=<*n*>] [ystart=<*n*>]
[length=<*n*:0.015>] [factor=<*n*:2.0>]
[color=<*c*:fore>] [isotropic=<*b*:false>]
[auto=<*b*:false>]
[omit=<*b*:false>] [frame=<*b*>]
[number=<*b*>] [tick=<*b*>]
[celsius=<*b*> | kelvin=<*b*> |
fahrenheit=<*b*> | rankin=<*b*>]

Restrictions: The **Arrhenius** statement must occur after any **Title** statements and before any **Plot**, **Annotate**, or **Key** statement. Automatic scaling (the "auto" keyword) assumes that data has have been read in by a previous **Read** statement.

Evaluator: No access to the expression evaluator through this statement.

Description:

The **Arrhenius** statement defines the plot axes for an Arrhenius plot. The temperature axis is along the top with the logarithmic (or "y") axis along the left side. The bottom axis is linear and corresponds to $1000/T_a$ where T_a is the absolute temperature. The units for the temperature scale may be specified as Kelvin, Rankin, degrees Celsius, or degrees Fahrenheit. Note that one does not specify the bottom (or "x-") axis min and max, but rather than the limits of the top or temperature-axis, tmin and tmax.

The **Arrhenius** statement is equivalent to three **Axis** statements followed by a **Use** statement. The following line, for example,

```
Arrhenius centigrade tmin=900.0 tmax=1200.0 ymin=0.1 ymax=1000.0
+         tdelta=50.0
```

is equivalent to the following statements:

```
Axis left log min=0.1 max=100.0
Axis top arrhenius centigrade min=900.0 max=1200.0 delta=50.0
Axis bottom linear
Use xaxis=top yaxis=left
```

4.2 Displaying Text

These statements are responsible for putting text on the plot. The **Title** statement is used to display title lines on the plot. The **Label** statement is used to label the plot axes. The **XLabel** and **Ylabel** statements are shorthand for labeling the bottom and left axes respectively. The **Annotate** and **Text** statements are used to display text within the area where data is plotted. Finally, plot keys can be generated using the **Key** and **Legend** statements. For all these statements, any text to be displayed that is enclosed by dollar signs ("$") will be evaluated before being displayed. For example, "$1.0+3.0/4.0$" will be displayed as "1.75".

4.2.1 The TITLE statement

Summary: The **Title** statement displays text centered at the top of the plot.

Format: **Title** [<*string*>]

Restrictions: The **Title** statement must occur before any axes are defined (via an **Axis**, **Linear**, **SemiLog**, **LogLog**, or **Arrhenius** statement) and before any labels are defined (via a **Label**, **XLabel**, or **YLabel** statement).

Evaluator: Any text to be displayed surrounded by dollar signs ($) will be evaluated before being displayed.

Description:

A title statement will display the title string centered at the top of the plot. Each successive title statement will appear on sequential lines on the plot. Any number of title lines are allowed;d, the plot area, however, shrinks to accomodate each additional title line.

4.2.2 The LABEL statement

Summary: The **Label** statement labels an entire axis or a specific tick on an already defined axis.

Format: **Label** [left=<*b*:false>] [right=<*b*:false>]
[bottom=<*b*:false>] [top=<*b*:false>]
[axis=<*b*:false> at=<*n*> [angle=<*n*:0>]]
[color=<*c*:foreground>] [text=<*c*>]

Restrictions: The **Label** statement must occur after any **Title** statement. If the "axis" keyword is specified, then the position along the axis ("at=") must also be specified, and the axis itself must have been previously defined.

Evaluator: Any text to be displayed surrounded by dollar signs ($) will be evaluated before being displayed.

Description:

The **Label** statement may be used to label one or more of the plot axes. The boolean keywords "left", "right", "bottom", and "top" are used to specify which axes are to be labeled; several axes may be labeled using one **Label** statement. The label text to be displayed is the value of the "text" keyword. The label text must be enclosed in quotes if it has any spaces in it. The label will appear in the foreground color unless specified with the "color" keyword. The label text usually applies to an entire axis, and the label text will appear centered along the specified axis.

The label text may, however, label only a specific tick on the axis. In this case, the boolean keyword "axis" is specified and the tick which will be labeled is given by the value of the "at" key word. The label text will appear on the axis in the place normally occupied by the tick numbering. The text will be displayed horizontally unless the "angle" keyword is used to specify a different angle. The following lines, for example,

```
Axis    bottom min=0 max=5 number=false
Label   bottom axis at=2 color=red text=two
Label   bottom axis at=4 color=red text=four
```

will generate an axis along the bottom with limits 0 and 5, but no numbers displayed along the bottom of the axis (though tick marks are displayed). The text "two" and "four" appears centered around the axis positions 2 and 4, respectively.

4.2.3 The XLABEL and YLABEL statements

Summary: Shorthand for specifying the text to appear below the bottom axis.

Format: **XLabel** <*string*>

Restrictions: The **XLabel** statement must occur after any **Title** statement.

Evaluator: Any text to be displayed surrounded by dollar signs ($) will be evaluated before being displayed.

Description:

The **XLabel** statement will display its argument as label for the entire bottom axis (the nominal x-axis). The string will be displayed in the foreground color. This statement is equivalent to a **Label** statement for the bottom axis. For example, the statement

```
XLabel Depth into Silicon (microns)
```

is equivalent to

```
Label bottom text="Depth into Silicon (microns)"
```

Summary: Shorthand for specifying the text to the left of the left vertical axis.

Format: **YLabel** <*string*>

Restrictions: The **YLabel** statement must occur after any **Title** statement.

Evaluator: Any text to be displayed surrounded by dollar signs ($) will be evaluated before being displayed.

Description:

The **YLabel** statement will display its argument as the label for the entire left axis (the nominal y-axis). The string will be displayed in the foreground color. This statement is equivalent to a **Label** statement for the left axis. For example, the statement

```
YLabel Arsenic Concentration (cm^-3)
```

is equivalent to

```
Label left text="Arsenic Concentration (cm^-3)"
```

4.2.4 The ANNOTATE and TEXT statement

Summary: The **Annotate** statement displays text within the area where the data is plotted.

Format: **Annotate** x=<*n*> y=<*n*>
[angle=<*n*:0>] [color=<*c*:fore>] [text=<*c*>]

Restrictions: Both horizontal and vertical axes must be defined before the **Key** statement.

Evaluator: Any text to be displayed surrounded by dollar signs ($) will be evaluated before being displayed.

Description:

The **Annotate** statement displays text within the data plotting area. The "x" and "y" keywords must be specified and are used to give the position where the text is to begin (i.e., the lower left corner). The annotation will be displayed horizontally (unless an angle, measured from the 12 O'clock position, is given by the "angle" keyword) and in the foreground color (unless overridden by the "color" keyword). Subsequent lines of the annotation are given in succeeding **Text** statements. The position of each annotation line is such that the successive line will not overlap. The "text" keyword itself is optional. If no text is specified by the **Annotate** statement, the first line of the annotation will be given by the first subsequent **Text** statement.

Summary: The **Text** statement gives subsequent lines in a multi-line annotation.

Format: **Text** [<*string*>]

Restrictions: Must occur after an **Annotate** statement.

Evaluator: Any text to be displayed surrounded by dollar signs ($) will be evaluated before being displayed.

Description:

This statement gives subsequent lines in a multi-line annotation. The color and angle are given in the most recent **Annotation** statement.

4.2.5 The KEY statement

Summary: The **Key** statement defines the location of the plot key.

Format: **Key** [x=<*n*>] [y=<*n*>]
[upper=<*b*:false> | lower=<*b*:false>]
[right=<*b*:false> | left=<*b*:false>]
[downward=<*b*>] [color=<*c*:fore>] [width=<*n*:16>]

Restrictions: Both horizontal and vertical axes must be defined before the **Annotate** statement. The **Key** statement must precede any **Legend** statements.

Evaluator: No access to the expression evaluator through this statement.

Description:

An identifying "legend" may be specified for each line to be plotted. The key statement specifies where the key is located, the legend statement describes the text to be displayed for the legend, and successive plot statements actually generate the legend lines. The start "x" and "y" position (along the axes currently in effect) may be explicitly specified, or the general location of the legend indicated using the upper, lower, right, and left positional boolean arguments. By default, the legend will appear in the center of the plot. Thus "Key left color=red" will cause the legend to begin at center left, with the legend text in red. The location of the legend determines the direction that the legend grows as additional entries are made; the legend grows downward if center or upper justified, and upward if defined to be a "lower" legend. This may be specifically set with the downward flag. The legend will also be right, center, or left justified if positioned along the left edge, center, or right edge respectively. The assumed width of legend text for justification is 16 characters; this may be increased using the "width" keyword.

4.2.6 The LEGEND statement

Summary: The **Legend** statement gives the text to displayed within the key.

Format: **Legend** *<string>*

Restrictions: The **Legend** statement must occur after a **Key** statement and **Read** statement and before the **Plot** statement that plots the curve the legend is for.

Evaluator: Any text to be displayed surrounded by dollar signs ($) will be evaluated before being displayed. For families of data the special variable `#family` will be equal to the family value. For contour data the special variable #contour will be equal to the contour value.

Description:

The text to appear with each legend statement. This text may include portions to be passed to the expression evaluator, delimited between dollar signs. For example, "`Legend VGS = $#family$`" will cause the family value during plotting to be substituted in the legend line. A new legend is generated for each succeeding plot statement (unless the "legend" keyword in the plot statement is set to false), and uses the same characteristics (linestyle, symbols, and colors) as in the plot statement.

4.3 Manipulating Data

4.3.1 The READ statement

Summary: The **Read** statement reads data into the program from a file.

Format: **Read** (file=<*c*> | stdin=<*b*:false> | comfile=<*b*:false>)
[xexp=<*c*:#1>] [yexp=<*c*:#2>]
[filter=<*c*>] [family=<*c*>]
[contour=<*c*> [non_uniform=<*b*:false>]
zmin=<*n*> zmax=<*n*> [deltaz=<*n*>]
[zlog=<*b*:false>]]
[numpoints=<*n*:0>] [format=<*c*:gdf>]
[sort=<*b*:false>] [verify=<*b*:false>]

Restrictions: The **Read** statement may occur anywhere in the command file.

Evaluator: The "xexp", "yexp", "family", and "contour" keywords all take values that are expressions.

Description:

The read statement allows one to read numeric data from a specified file, from the *standard input*, or from the command file itself. The format of the data file as well as the expression evaluator is described in sections 5 and 6 below. By default, the x values will be taken from the first column and the y from the second column of the data file. The verify flag will cause GIRAPHE to write to *standard error* (stderr) the x and y values as they are read from the file. Optionally, the values may be sorted (on the x values) after they are read in. GIRAPHE has a default limit on the number of data points one may read in (currently 400 points); if one has more data than this, the maximum number of points should be specified.

As the data is read in, it may be "filtered" through the expression evaluator. Only those points satisfying the filter condition are stored in the x and y arrays of values to be plotted. Each data point may also have a z value (or a "family" value) associated with it. This is of use in the plotting of families of curves. For instance, MOS current-voltage curves are usually parameterized in terms of the gate voltage. A read statement of form "Read file=foo x=VDS y=ID family=VGS" will cause distinct lines to be drawn each time the value of the family variable (VGS here) changes.

GIRAPHE also reads data to be plotted using iso-value contours. In this case, the column (or variable) in the datafile containing the z values is specified with the "contour" keyword, along with the minimum and maximum contour values to be plotted. The increment between contours may be specified ($delta_i$); if the contour is logarithmic this is a multiplicative increment, otherwise it is an additive increment.

4.3.2 The PLOT statement

Summary: The **Plot** statement plots data previously read in.

Format: **Plot** [symbol=<*c*>] [linestyle=<*c*>] [fill=<*c*>] [color=<*c*:next>] [label=<*c*>] [legend=<*b*:true>] [closed=<*b*:false>] [curve=<*b*:false>] [polygon=<*b*:false>] [outline=<*b*:false>] [order=<*c*:symbol:color:fill:line>]

Restrictions: Both horizontal and vertical axes must be defined before the **Plot** statement. Data must have been previously read in using a **Read** statement.

Evaluator: No access to the expression evaluator.

Description:

Once the data has been generated using a read statement, it may be plotted using one or more **Plot** statements. The data may be plotted as a set of isolated symbols, as a curve, as a polygon, or any combination of these. The possible symbols are "square", "circle", "triangle", "box", "bullet", "wedge", "plus", and "cross", or any alphanumeric character. In the case of curves, the linestyle is solid by default. The possible linestyles are "solid", "dashed", "dotted" and "ddotted" (dash-dotted). In each of these, the word "next" may be used, so that the symbol, color, linestyle, or fill patterns will be cycled through. Polygons may be outlined in the current linestyle and color, with a specified fill pattern. Currently the only well-defined fill pattern is "solid". The color of the symbols, curves, or polygons are specified with the color argument.

In the case of family or contour plots, successive lines can be plotted with different characteristics. In place of a single value to the symbol, linestyle, and color arguments, multiple values may be specified, and are separated by colons (for example `symbol=box:wedge:bullet`"). Since a large number of families are possible, the characteristics of symbols, lines, and polygon fill colors are not stepped in unison. Rather, each different symbol specified will be drawn first, then the color changed and each symbol cycled through again. The order in which this cycling occurs may be specified with the order parameter.

If a key and legend statement have been previously issued, a legend line will by default be written for each new plot statement (or family curve within a single plot statement). This legend may be suppressed by setting the "legend" keyword in the plot statement to false.

4.3.3 The WRITE statement

Summary: The **Write** statement writes the internal data to a file. This file may subsequently read in using a **Read** statement.

Format: **Write** (file=<*c*> | stdout=<*b*:false>) [format=<*c*:gdf>]
[xexp=<*c*:#x>] [yexp=<*c*:#y>] [zexp=<*c*:#z>]
[xname=<*c*>] [yname=<*c*>] [zname=<*c*>]

Restrictions: The **Write** statement must occur after a **Read** statement.

Evaluator: The values of the "xexp", "yexp" and "zexp" keywords are expressions that are evaluated by the expression evaluator.

Description:

Once data has been read in using the expression evaluator, as well as possibly filtered or contoured, one may write the reduced data back out to a named file. The x, y, and z value names may be specified (column names), as well as the x, y and z values to be output.

4.4 Miscellaneous Statements

4.4.1 The COMMENT statement

Summary: Used for commenting the command file.

Format: **Comment** [<*string*>]

Restrictions: None.

Evaluator: No access to the expression evaluator.

Description:

The Comment statement is not executed, but allows for documentation of the command file.

4.4.2 The SYNTAX statement

Summary: Generates a synopsis of the given statement's syntax.

Format: **Syntax** <*string*>

Restrictions: None, although only useful when running GIRAPHE as an interactive shell.

Evaluator: No access to the expression evaluator.

Description:

Causes the above syntax to be output for the specified syntax. While not as comprehensive as online help, this does summarize the various parameters for the requested statement.

4.4.3 The SET statement

Summary: Set data file parameter values from command file.

Format: **Set** [<*string*>]

Restrictions: None.

Evaluator: String is processed by the expression evaluator.

Description:

Allows one to set data file parameter values from the command file. See the discussion of the expression evaluator in Section 6.

4.4.4 The INCLUDE statement

Summary: Reads data or command statements from another source.

Format: **Include** (File=<*c*> | stdin=<*b*:false>) [pluscontinue=<*b*:true>] [dataonly=<*b*:false>] [library=<*b*:false>]

Restrictions: None.

Evaluator: No access to the expression evaluator.

Description:

The include statement allows one to textually include, either from a file or from the *standard input*, additional command file statements. By default, these statements may be continued with a "+" on succeeding lines. The dataonly flag indicates that the information will not have *giraphe3* statements, but only data.

4.4.5 The END statement

Summary: Terminates the program.

Format: **End** <*string*>

Restrictions: None.

Evaluator: No access to the expression evaluator.

Description:

Completes drawing of any pending data or axes, and terminates the program. The string argument to end is used only for documentation.

5 Data Format

GIRAPHE is fundamentally a scientific data plotting program; the format of the data is described in this section. The data may be provided to the program in three ways: as a separate data file, as part of the GIRAPHE command file, or from the standard input. The data format is identical in these three cases (although a terminating ".end" may be omitted from a separate data file).

5.1 Basic format

The basic format is tabular in nature. Each distinct data point in the GIRAPHE data format (sometimes referred to as "gdf") is formed by a row of numbers in the data file. The values in each row are separated into one or more columns of numbers; each number may take any of the usual printed representation of floating point numbers, with one important exception. No row should have as its first column a number that begins with a decimal point (i.e. ".49"); the period indicates that a data file directive is to follow, so that the number should be represented with a preceding zero (i.e. "0.49") instead. The data columns may be referred to in the GIRAPHE command file by names such as "#1" or "#3". Examples of the use of the GIRAPHE data format can be found in Section 2.

5.2 Data directives

In its simplest form, then, a GIRAPHE data file consists solely of the x and y values to be plotted. A number of data directives are provided to simplify specification and use of data. Each of these begin with a period in the first position on a line, and are described below.

5.2.1 .column

Columns may be labeled with the ".column" directive, followed on the same line by the names of the successive columns in the data. Once labeled, the values in these columns may be referred to by name in the GIRAPHE command file when specifying the x, y, and family values to be plotted.

5.2.2 .parameter

While the ".column" directive provides a way of associating a name with a column, the ".parameter" and ".set" directives enable one to define additional names and values without specifying numeric values in the columns. The ".parameter" directive, followed by a list of names on the same line, defines these names.

5.2.3 .set

After a ".parameter" definition, subsequent ".set" directives assign values to these parameters. Each ".set" should be followed by one or more <parameter=value> pairs. The ".set" directive may appear anywhere in the data file, and sets the value of the parameter for the rest of the data file or until superseded by another ".set". For instance, the parameter "vt" is declared with ".par vt", and can be set anywhere later in the file with ".set vt=1.45", and still later by ".set vt=1.55". The use of a parameter is equivalent to a column by the parameter name with all entries equal to the set value.

5.2.4 .remark

Lines in the data file beginning with ".remark" are ignored by GIRAPHE. These remarks, combined with column specifications, provide documentation about the data itself. Lines with nothing but white space on them are ignored by GIRAPHE, and can be used in the data file.

5.2.5 .on and .off

The ".off" directive causes subsequent data lines to be ignored by GIRAPHE until a line beginning with ".on" is encountered, at which time GIRAPHE resumes processing lines as data. Thus, sections of the data file that are not to be interpreted by GIRAPHE as data may be enclosed between lines beginning with ".off" and ending with ".on".

5.2.6 .end

The ".end" directive indicates the end of data. When data is coming from a file, the ".end" may be omitted, as the end of the file implicitly defines the end of the data. When the data is interposed in the GIRAPHE command file itself, or is coming from the standard input, the ".end" must be present so that GIRAPHE knows to return to interpreting lines as GIRAPHE statements rather than data.

Table III: Mathematical Functions

Function	Description
`abs`	Absolute value
`acos`	Inverse cosine
`asin`	Inverse sine
`atan`	Inverse tangent
`ceil`	Ceiling (nearest integer greater than or equal to number)
`cos`	Cosine
`cosh`	Hyperbolic cosine
`cot`	Cotangent
`erf`	Error function
`erfc`	Complementary error function
`exp`	e raised to power
`exp10`	10 raised to power
`floor`	Floor (nearest integer less or equal to number)
`log`	log base e
`log10`	log base 10
`print`	Print value to stdout
`sin`	Sine
`sinh`	Hyperbolic sine
`tan`	Tangent
`tanh`	Hyperbolic tangent

6 Expression Evaluator

A powerful expression evaluator is included in GIRAPHE to make it easy to deal with large amounts of data, as well as to perform some amount of manipulation of raw data before plotting. The evaluator provides the usual infix operations, including "+", "-", "+", "*", "/", "**" or "^" for exponentiation, and "%" for modulo. In addition, a number of predefined mathematical functions are provided. These functions, used as "log(x)", are summarized in Table III. Finally, boolean operations including "^" (unary negation), "^=" (not equal), "<=", ">=", "<>", ">", "<", "==", "&&", "||", and "=" are provided. These are supplemented by predefined constants, as defined in Table IV.

The precedence of calculations using these operators is to perform operations nested within parentheses first (from the inside toward the outside), followed by any of the functions with parenthesized notations (such as "atan(x)"). Precedence then goes to the other unary operators ("-", "+", and "^"), followed by exponentiation, then successively binary arithmetic operators ("*" and "/", then "+" and "-"),

Table IV: Predefined Constants

Constant	Value	Description
#pi	3.14159265358979323846	
#e	2.71828182845904523536	
#gamma	0.57721566490153286061	Euler's constant
#phi	1.61803398874989484820	Golden ratio
#q	1.60219E-19	electric charge
#h	6.62620E-34	Plank's constant
#hbar	1.05459E-34	hbar
#NAvg	6.02217E23	Avogadro's number
#me	9.10956E-31	electron mass
#kboltz	1.38062E-23	Boltzman constant
#c	2.997925E08	speed of light
#true	1.0	true
#false	0.0	false

arithmetic comparisons ("<=", ">=", "<>", ">" and "<"), conditional tests ("^=" and "=="), the boolean "and" operator ("&&"), the boolean "or" ("||"), and finally assignment ("=").

The expression evaluator is available for two kinds of use in GIRAPHE. The first of these is in calculating or extracting values to be put into the x, y, and z (or family) arrays for plotting. Here, one typically reads some column or combination of columns, and performs numerical calculations before assignment to the "xexp", "yexp", or "family" values in the read statement. For instance, the argument "xexp=#1+log(vt*1.1)" would assign x with the value of column one added to the log of one plus whatever value (either a parameter or a column) vt holds.

The second primary use is to "filter" the specified data. For instance, one may only want to plot points where vt is between 1 and 2. In this case, the boolean parameters are used: "filter=vt>=1&&vt<=2".

The expression evaluator is also invoked by any GIRAPHE statements that put text on the plot, allowing calculation and presentation of numeric (or resolution of named variable) values. The statements where this is possible include the "Label", "XLabel", "YLabel", "Title", "Legend", and "Annotate" statements.

Index

NAME

giraphe3 – scientific plotting program

SYNOPSIS

giraphe3 [giraphe3-inputfile] < **-t** *mfb*-devicetype > [**-b** background-color] [**-f** foreground-color] [**-d** device-name] [**-l** log-file] [**-o** < output-file >] [**-q** queue-name] [**-i**] [**-w**] [**-+**] [**-s**]

DESCRIPTION

Giraphe3 generates scientific plots using an input command file or direct user input and any number of data files. *Giraphe3* produces output for a number of different graphics devices, including both graphics terminals and hard copy devices. The *giraphe3* command file (normally the first parameter of the *giraphe3* command) defines the characteristics of the desired plot, as well as specifies the data to be plotted. The format of both the *giraphe3* input file and the *giraphe3* data files are discussed below.

If the command file argument is omitted and the **-** option is used, *giraphe3* will use standard input rather than a command file as the source of statements to execute. The program can be also be run as an interactive "shell" (where the **-+** option is recommended for interactive use). *Giraphe3* statements allow one to incorporate data directly into the command file, or to read the standard input for data. The program can thus serve as a scientific plotting filter between data generating programs and display devices.

Giraphe3 is usually invoked with the **-t** option, followed by the *device type*. If the **-t** option is omitted, *giraphe3* will try to figure out the mfb device type by itself. If the **-q** has been specified, it uses the queue name to figure out the device type. If **-q** is not specified, *giraphe3* will use the environment variable "MFBTERM" if set. Failing that, the "DISPLAY" environment variable is checked, and if set *giraphe3* will use "X" as the display type. In specifying the device type explicity with **-t**, the device type must be a valid *mfb* device type; under the X Window System, a window may be created with the "X" device type. The DISPLAY environment variable is used to determine which physical display device to use. Sample device names include

d3 VT241 terminal.

t25 Tektronix 4125 terminal.

x Creates an "mfb" emulation window. The user must specify the location of this window with the mouse. Note that automatic refresh of this window does NOT occur, so that moving, iconifying, or uncovering the window will result in destruction of the plot.

hc HPGL (Hewlett Packard Graphics Language) file output.

ps PostScript format file output. The **ps** format will print in the lower half of a page in a size appropriate for inclusion in a paper or report. Other device names with similar results include **ps2** which will produce a full page PostScript plot, and **ps3** which produces a full page plot, with larger lines and text. **ps4** will generate a full horizontal rather than vertical page.

pp Unix plot(5) format file output. This file can then be passed through the standard unix plot filters. For instance, the "pp" file generated by *giraphe3* can be displayed on an ascii display using "plot -tcrt file.pp".

The other options are used as follows:

-d Changes the device where the plot is to be displayed. By default, the plot is written to the device where the command is generated. The **-d** option can be used to redirect output to another device. Under the X window system, the plot will be directed to the named display (i.e. garbo:0).

- **-b** Sets the background color to that specified. Default is black, except for file output which will use white as the background color.
- **-f** Sets the foreground color to that specified. Default is white, except for file output which which will use black as the foreground color.
- **-o** Specifies that output is to go to the specified file rather than to standard output. By default, the name of the output file will be the name of the command file, with the device type replacing the extension on the command file name. For instance, "foo.grp -t ps4 -o" will produce "foo.ps4" as output. If an argument (which is not another dashed option) immediately follows the **-o** option, then the output file will have that as the file name.
- **-w** If the graphics device has a pointing device, the plot will remain until the user types a point (or hits a key) before terminating. If no graphics device is available, waits for the user to enter ctrl-D. By default the **-w** option is in effect for X window display.
- **-q** The hardcopy output file will be queued to the specified printer by *Giraphe3* upon completion.
- **-s** *Giraphe3* performs silently. The user is not told the version number or when the plot is done; normally this information is directed to the standard output. Error messages are still written to the standard output.
- **-** Reads command statements from standard input rather than from s specified file.
- **-+** Reads command statements from standard input rather than from s specified file; but without command line *continuations*. Normally, a "+ " as the first character on a command line causes that command line to be considered a continuation of the previous textual line. When running *Giraphe3* as an interactive shell, however, one normally wants a command to be executed immediately, necessitating the **-+** option.
- **-i** Inhibits the initizialization of the color map. If **-i** is used, the colors already defined for the device (if there are any) will be used instead. Particularly useful when generating output for VT241's.
- **-l** A log file of errors (or verification output if so requested inside the *giraphe3* input file).

COMMAND FILE FORMAT

The *giraphe3* command file consists of a sequence of statements, each statement on a new line. A statement may be continued to the following line with a "+ " as the first character of the continuation line. Some of the statements must be issued before others; these are mentioned for each of the statements below.

Each statement may take a number of named arguments to which values may be assigned. These arguments may be mandatory (enclosed by "( )") or optional (enclosed by "[ ]"). Occasionally only one parameter of several may be chosen (these "pick one" arguments are separated by "|"). The arguments may have values of several types. String arguments are indicated by < s> , and may have imbedded spaces. Character arguments are indicated below by < c> , and must not have imbedded white space (since white space separates arguments in the statement), unless the entire value is enclosed in double quotes. Numeric values are indicated by < n> . Boolean values are denoted below with a < b> type. Note that the simple presence of the boolean argument in a statement indicates "true", so that "Label left" is equivalent to "Label left= true". To set a boolean to false one may use the up-arrow, as in "Label ^left", or one may use "Label left= false". Finally the default values of each argument, if any is defined, are shown following the type specifier and a colon.

Title [< s>]

A title statement will display the title string centered at the top of the plot. Each

successive title statement will appear on sequential lines on the plot. While any number of title lines are allowed, all title statements must appear before the axes are defined.

Comment
[< s>]

The comment statement is not executed, but allows for documentation of the command file.

Axis [left= < b:false>] [right= < b:false>]
[top= < b:false>] [bottom= < b:false>]
((min= < n> max= < n>) |auto= < b:false>)
[start= < n>] [delta= < n>] [freq= < n>]
[length= < n:0.015>] [factor= < n:2.0>]
[color= < n:foreground>]
(linear= < b> |logarithmic= < b> |
(arrhenius= < b> [celsius= < b> |kelvin= < b> |
farenheit= < b> |rankin= < b>]))
[omit= < b:false>] [frame= < b>] [number= < b>] [tick= < b>]

The axis statement allows one to define a single axis along the top or bottom, or along the left or right edge of the plot. One must either specify the minimum axis value (actually the left-most or bottom-most value) and maximum (the right-most or top-most) value, or indicate that the program is to determine these values. If one is to use the automatic axis scaling, however, the axis statement must follow an earlier read statement.

A number of parameters give control over the position and appearance of the major and minor tick marks along the axis. The start parameter indicates the value to begin labeling as a major tick. The delta parameter is the increment between major ticks, and the freq indicates how many minor tick spaces lie between each major tick. The length parameter gives the length of the minor tick in percentage of the plot area on the screen; the major tick length is this length multiplies by the factor parameter. For normal use, specification of the minimum and maximum axis values is sufficient; the program is proficient at choosing major and minor tick marks. By default, the axis and tick color will be the foreground color, this may be set with the color parameter. The tick marks may be suppressed by setting the tick parameter to false, and setting the number parameter false suppresses numbering of the axis.

The character of the axis must also be specified. The axis may be linear or logarithmic, or may be an arrhenius scale (in celsius, kelvin, farenheit, or rankin). The axis may be omitted altogether by setting omit to true. Setting the frame parameter true will cause a box to be drawn around the entire plot. Note that one may specify different axes for the top and bottom of the plot; if one does not specify one or the other, the axis and tick marks will be redrawn on along the missing edge at the completion of the plot.

No title statements may follow any axis statement. Furthermore, all plot, key, and annotate statements should follow the axes definitions.

Linear [xmin= < n:0.0>] [xmax= < n:100.0>]
[ymin= < n:0.0>] [ymax= < n:100.0>]
[xdelta= < n>] [ydelta= < n>]
[xfreq= < n>] [yfreq= < n>]

[xstart= < n>] [ystart= < n>]
[length= < n:0.015>] [factor= < n:2.0>]
[color= < c:fore>]
[omit= < b:false>] [frame= < b>] [number= < b>] [tick= < b>]
[isotropic= < b:false>]

This is a short hand statement for specifying a linear plot (both left and bottom axes are linear). Here both the x (horizontal) and y (vertical) axis limits must be specified. Note that a decreasing axis is possible, as is "linear xmin= 100 xmax= 0". The isotropic parameter indicates that one unit in x is equal to one unit in y, and that the plot area should be scaled to as to maintain a one-to-one aspect ratio. Other parameters are identical to those in the more general axis statement.

SemiLog

Same parameters as for the Linear statement.

This is a short hand statement for specifying a semilog plot, where the x (horizontal) axis is assumed to be linear, and the y (vertical) axis is assumed to be logarithmic. Other parameters are identical to those in the more general axis statement.

LogLog

Same parameters as for the Linear statement.

This is a short hand statement for specifying a log-log plot, where both the x (horizontal) and y (vertical) axes are logarithmic. Other parameters are identical to those in the more general axis statement.

Arrhenius

[tmin= < n:900.0>] [tmax= < n:1200.0>]
[ymin= < n:1.0>] [ymax= < n:100.0>]
[tdelta= < n>] [ydelta= < n>]
[tfreq= < n>] [yfreq= < n>]
[tstart= < n>] [ystart= < n>]
[length= < n:0.015>] [factor= < n:2.0>]
[color= < c:fore>]
[omit= < b:false>] [frame= < b>] [number= < b>] [tick= < b>]
[celsius= < b> |kelvin= < b> |
farenheit= < b> |rankin= < b>]
[isotropic= < b:false>]

A set of arrhenius axes are defined, with the temperature along the bottom and the y value along the side. The units for the temperature scale may be specified as celsius, kelvin, farenheit, or rankin. Note that one does not specify the "x" axis min and max, but rather the "temperature" tmin and tmax.

Use [xaxis= < c>] [yaxis= < c>]

By default, the x axis for actual plotting is the horizontal bottom axis, and the y values are defined along the vertical left axis. One may specify the x or y axes as "left", "bottom", "right", or "top" to correspond previously defined axis values. This statement must follow those axes definitions.

Label [left= < b:false>] [right= < b:false>]
[bottom= < b:false>] [top= < b:false>]
[axis at= < n> [angle= < n:0>]]
[color= < c:foreground>] [text= < c>]

The Label statement defines the text to appear along the left, bottom, right, or top axis. The color of this text may be defined, by default the foreground color is used. The text should be enclosed in quotes if it has any tabs or spaces in it; this text will appear centered below or outside the axis (and below the axis numbering).

In addition, one may specify text to appear along the axis in the position normally occuppied by the axis numbering by using the axis argument. In this case, one must specify the axis value (and optionally an angle) at which the text is to appear.

XLabel

< s>

Shorthand for specifying the text to appear below the x axis.

YLabel

< s> Shorthand for specifying the text to appear below the y axis.

Annotate

x= < n:1.0> y= < n:1.4e+ 04> [angle= < n:0>]
[sizehor= < n>] [sizever= < n>]
[color= < c:fore>] [text= < c>]

The annotate statement enables one to place text within the data plotting area itself. One must specify the x and y position for the text to begin at, and may optionally provide an angle for the writing of the text. The color of the text may also be specified. The sizehor and sizever arguments are not yet implemented. The text argument is optional. Instead, one may use later Text statements to write text to the specified location specified in the closest preceeding annotate statement. Successive text statements will write incrementally offset annotations. Annotations must be preceded by the definition of the axes.

Text [< s>]

See Annotate statement.

Read (file= < c> |stdin= < b:false> |comfile= < b:false>)
[xexp= < c:# 1>] [yexp= < c:# 2>]
[verify= < b:false>] [filter= < c>]
[family= < c>] [sort= < b:false>]
[contour= < c> zmin= < n> zmax= < n> [deltaz= < n>]
[non_uniform= < b:false>] [zlog= < b:false>]
[cycle= < n:1>]]
[numpoints= < n:0>] [format= < c:gdf>]

The read statement allows one to read numeric data from a specified file, from the standard input, or from the command file itself. If the data is to come from the command file, it must appear immediately following the read statement and before any other command file statements; note also that the data must end with ".end" in order for *giraphe3* to return to executing statements rather than continue to read lines as data from the command file. The format of the data file as well as the expression evaluator is is described further below. By default, the x values will be taken from the first column and the y from the second column of the data file. The verify flag will cause *giraphe3* to write to stdout the x and y values as they are read from the file. Optionally, the values may be sorted (on the x values) after they are read in. *Giraphe3* has a default limit on the number of data points one may read in (currently 400 points); if one has more data than this, the maximum number of points should be specified.

As the data is read in, it may be "filtered" through the expression evaluator. Only those points satisfying the filter condition are stored in the x and y arrays of values to be plotted. Each data point may also have a z value (or a "family" value) associated with it. This is of use in the plotting of families of curves. For instance, MOS IV curves are usually parameterized in terms of the gate voltage. A read statement of form "read file= foo x= VDS y= ID family= VGS" will cause distinct lines to be drawn each time the value of the family variable (VGS here) changes.

Giraphe3 can also read data to be plotted using iso-value contours. In this case, the column (or variable) in the datafile containing the z values is specified, along with the minimum and maximum contour value to be plotted. The increment between contours may also be specified; if the contour is logarthmic this is a multiplicative increment, otherwise it is an additive increment. If the contour data does not form a rectilinear array in x and y, the non_uniform flag may be set true, in which case *giraphe3* will interpolate random data points onto a uniform grid for contour plotting. This non_uniform option has not yet been implemented.

Plot [symbol= < c>]
[curve= < b:false>] [linestyle= < c>]
[polygon= < b:false>] [fill= < c>]
[closed= < b:false>] [outline= < b:false>]
[color= < c:next>] [order= < c:symbol:color:fill:line>]
[label= < c>] [legend= < b:true>]
[major= < b:false>] [minor= < b:false>]

Once the data has been generated using a read statement, it may be plotting using one or more plot statements. The data may be plotted as isolated symbols, curves, or polygons, or as any combination of these. The possible symbols are "square", "circle", "triangle", "box", "bullet", "wedge", "plus", and "cross", or any alphanumeric character. In the case of curves, the linestyle is solid be default. The possible linestyles are "solid", "dashed", "dotted" and "ddotted" (dash-dotted). In each of these, the work "next" may be used, so that the symbol, color, linestyle, or fill patterns will be cycled through. In the case of polygons, the polyons may be outlined in the current linestyle and color, with a specified fill pattern. Currently the only well defined fill pattern is "solid". The color of the symbols, curves, or polygons are specified with the color argument.

In the case of family or contour plots, successive lines can be specified to be plotted with different characteristics. In place of a single value to the symbol, linestyle, and color arguments, a multiple number of values may be specified separated by colons (for example "symbol= box:wedge:bullet"). Since a large number of families are possible, the characteristics of symbols, lines, and polygon fill colors are not stepped in unison. Rather, each different symbol specified will be drawn first, then the color changed and each symbol cycled through again. The order in which this cycling occurs may be modified with the order parameter.

If a key statement has been previously issued, a legend line will by default be written for each new curve. This may be suppressed with by setting the legend boolean argument false.

Key [x= < n>] [y= < n>]
[upper= < b:false>] [lower= < b:false>]
[right= < b:false>] [left= < b:false>]
[downward= < b>] [color= < c:fore>]
[width= < n:16>]

An identifying "legend" may be specified for each line to be plotted. The key statement specified where the key is to located, the legend statement describes the text to be displayed for the legend, and successive plot statements actually generate the legend lines. The start x and y position may be specified, or the general location of the legend using the upper, lower, right, and left positional boolean arguments. By default, the legend will appear in the center of the plot. Thus "Key left color= red" will cause the legend to begin at center left, with the legend text in red. The location of the legend determines the direction that the legend grows as additional entries are made; the legend grows downward if center or upper justified, and upward if defined to be a "lower" legend. This may be specifically set with the downward flag. The legend will also be right, center, or left justified if positioned along the left edge, center, or right edge respectively. The assumed width of legend text for justification is 16 characters; this may be modified if more is needed.

Legend < s>

The text to appear with each legend statement. This text may include portions to be passed to the expression evaluator, delimited between dollar signs. For example, "Legend VGS = $# family$" will cause the family value during plotting to be substituted in the legend line.

Set < s>

Allows one to set data file parameter values from the command file. See discussion of the expression evaluator below.

Syntax < s>

Causes the above syntax to be output for the specified syntax. While not as comprehensive as online help, this does summarize the various parameters for the requested statement.

Write file= < c> [format= < c:gdf>]
[xexp= < c:# x>] [yexp= < c:# y>] [zexp= < c:# z>]
[xname= < c>] [yname= < c>] [zname= < c>]

This statement has not been implemented yet. Once data has been read in using the expression evaluator, as well as possibly filtered or contoured, one may write the reduced data back out to a named file. The x, y, and z value names may be specified (column names), as well as the x, y and z values to be output.

Include
(File= < c> |stdin= < b:false>) [PlusContinue= < b:true>]
[Dataonly= < b:false>] [Library= < b:false>]

The include statement allows one to textually include, either from a file or from the standard input, additional command file statements. By default, these statements may be continued with a "+ " on succeeding lines. The dataonly flag indicates that the information will not have *giraphe3* statements, but only data.

End < s>

Terminates the program.

The *giraphe3* syntax file used may be changed by setting a variable "Giraphe3Syntax" in the environment.

DATA FILE FORMAT

The data file should contain data points (in ascii) in columnar format. These columns may be referred to in the *giraphe3* command file by names such as "# 1" or "# 3". Alternatively, one may label the columns inside the data file, with a line beginning ".col" followed on that line by the names of the respective columns. Names of parameters may also be defined with a ".par" line, followed by the names of parameters (all parameters must be defined on the same .par line). The values of these parameters are set with ".set", followed by the parameter name, an equal character, and its value; this is equivalent to a column by that name with all entries equal to the value. For instance, the parameter "vt" is declared with ".par vt", and can be set anywhere later in the file with ".set vt= 1.45". Comment lines may be inserted anywhere in the data file, with a ".remark" at the beginning of the line.

EXPRESSION EVALUATOR

A powerful expression evaluator is included in *giraphe3* to make it easy to deal with large amounts of data, as well as to perform some amount of data reduction on raw data before plotting. The evaluator provides infix notation operations that one is used to, include "+ ", "*", "-", "/", "**" or "^" for exponentiation, and "%" for modulo. In addition, a number of predefined mathematical functions are provided (see 3m man pages for descriptions of these functions). These functions, used as "log(x)", include "abs", "acos", "asin", "atan", "ceil", "cosh", "cos", "cot", "erfc", "erf", "exp10", "exp", "floor", "log10", "log", "sinh", "sin", "sqrt", "tanh", "tan", and "print". Finally, boolean operations including "!", "!= ", "^", "< = ", "> = ", "< > ", "> ", "< ", "= = ", "&&", "|", and "= " are provided. These are supplemented by a number of predefined constants, including "# pi", "# e", "# gamma", "# phi", "# q", "# h", "# hbar", "# NAvg", "# me", "# kboltz", "# c", "# true", and "# false".

The expression evaluator is available for two kinds of use in *giraphe3*. The first of these is in calculating or extracting values to be put into the x, y, and z (or family) arrays for plotting. Here, one typically reads some column or combination of columns, and performs numerical calculations before they are assigned to the "xexp", "yexp", or "family" values in the read statement. For instance, the argument "xexp= # 1+ log(vt*1.1)" would assign x with the value of column one added to the log of one plus whatever value (either a parameter or a column") vt holds.

The second primary use is to "filter" the specified data. For instance, one may only want to plot points where vt is between 1 and 2. In this case, the boolean parameters are used to write, "filter= vt> = 1&&vt< = 2".

FILES

/cad/lib/giraphe3.syn

SEE ALSO

Example *giraphe3* command files can be found in **/cad/src/giraphe3/examples.**

For high quality hard copy plots (limited to PostScript output), the plotting package of Merit Hung is of interest. For manipulation of "tables" of numeric data, see the graph+ routines.

AUTHORS

Robert Harris and Duane Boning, Massachusetts Institute of Technology

BUGS

The parameters on input lines must be reasonable, or program may bomb.

The program currently demands that the command file have the ".grp" extension.

Warning: With postscript output, an error may cause no page to be printed.

The error messages generated by the program are usually cryptic.

Program occasionally core dumps with no good error message. When this happens, the terminal is sometimes left in raw mode (because of MFB), even if output was going to a file.

It's a new program, so there may be many unknown bugs. Bug reports should include the *giraphe3* version number, and as much as you can tell about the problem. Address reports to bug-giraphe@bacall.mit.edu.